Proof of translation in natural semantics

Joelle Despeyroux

INRIA

CENTRE
SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France

Tél. (3) 954 90 20

Rapports de Recherche

N° 514

PROOF OF TRANSLATION IN NATURAL SEMANTICS

Joëlle DESPEYROUX

Avril 1986

PROOF OF TRANSLATION IN NATURAL SEMANTICS

Joëlle Despeyroux

INRIA, Sophia-Antipolis
Route des Lucioles, 06565 Valbonne Cedex, France

Abstract

We have retained the purely 'formal system' part of the structural operational semantics 'a la Plotkin', and have named that view of it 'natural semantics'. Numerous examples written in this semantics have been entered in the meta-compiler Mentor + Typol. We show here that natural semantics enables us in a single formalism, to define dynamic semantics and translation of languages and to prove the correctness of a translation. Our criterion of correctness is the validity of two inference rules in a theory. Proofs are made by induction on the length of the proof. We illustrate the method on an example, treated in full: translation from Mini-ML into CAM (Categorical Abstract Machine).

Résumé

Nous avons retenu la partie purement "système formel" de la sémantique operationelle structurelle 'a la Plotkin', et avons appellé cette vue de celle-ci "sémantique naturelle". De nombreux exemples ecrits en sémantique naturelle ont ete entrés dans le méta-compilateur Mentor + Typol. Nous montrons ici que la sémantique naturelle nous permet dans un seul formalisme, de définir des sémantiques dynamiques et des traductions et de prouver la correction d'une traduction. Notre critère de correction est la validité de deux règles d'inference dans une théorie. Les preuves sont faites par induction sur la longueur de la preuve. Nous illustrons la méthode sur un exemple, complètement traité: la traduction de Mini-ML dans la CAM (Categorical Abstract Machine).

PROOF OF TRANSLATION IN NATURAL SEMANTICS

Joëlle Despeyroux

INRIA, Sophia-Antipolis
Route des Lucioles, 06565 Valbonne Cedex, France

Abstract

We have retained the purely 'formal system' part of the structural operational semantics 'a la Plotkin', and have named that view of it 'natural semantics'. Numerous examples written in this semantics have been entered in our meta-compiler Mentor + Typol. We show here that natural semantics enables us in a single formalism, to define dynamic semantics and translation of languages and to prove the correctness of a translation. Our criterion of correctness is the validity of two inference rules in a theory. Proofs are made by induction on the length of the proof. We illustrate the method on an example, treated in full: translation from Mini-ML into CAM (Categorical Abstract Machine).

1. Introduction

1.1. Natural Semantics

The natural semantics of a language is given by a formal system (a set of axioms and inference rules) which defines a set of valid theorem (a theory). Theorems of interest are, for example:

$\vdash P : \alpha$	P executes to α
$\vdash P \in \pi$	type of P is π
$\vdash P \rightarrow P'$	P translates to P'

The name *natural* comes from the fact that this system is given in the Gentzen's system style [5][1], in which we can make *natural deduction* [1] [17] [6]. Furthermore, semantics written in this style appears to be rather intuitive, so that *natural* may also be understood in the lay-man's sense. Natural semantics has its origin in the structural operational semantics most fully developed in [15]. But we focus on the pure logical part of it. Note that natural semantics is not intrinsically operational (for us "$\rightarrow$" simply denotes a predicate, and not a transition of an abstract machine), and can even be non structural (see later on the -usual- semantics of application in ML).

[1] We use *natural deduction* in its first meaning in that we don't always write an introduction rule and an elimination rule for each constructor, so we don't have a notion of *normal proof*, in dynamic semantics at least.

This work is partially supported under ESPRIT p.348

Numerous examples, in static semantics, dynamic semantics and translation, have been written in natural semantics and are presented in [7]. All these examples have been compiled and so have mechanically generated running type-checkers, interpreters and translators (compilers). The programming language supporting natural semantics is called (for historical reasons) Typol. The development of Typol is an extension of the work on the syntactic meta-editor Mentor [14]. It is not our purpose here to describe this language [3][4]. It is sufficient to say that a Typol program is a first order logic whose terms are abstract syntax trees (which may be graphs as we shall see later on).

1.2. Proof of translation

Our purpose here is to show that natural semantics enables us (in a single formalism) to define dynamic semantics and translations, and to prove the correctness of these translations. Let's recall the usual diagram L. Morris:

$$\begin{array}{ccc} L_1 & \xrightarrow{T} & L_2 \\ {\scriptstyle 1}\downarrow & & \downarrow{\scriptstyle 2} \\ SD(L_1) & \xrightarrow[t]{} & SD(L_2) \end{array}$$

where the dynamic semantics of L_i is given by a semantic domain SD(L_i) and a semantic mapping L_i_DS from obkects (programs) of L_i into values of SD(L_i). The mapping T is the translation of programs and t is the translation of semantics values. In our context semantic domains are integers, booleans, closures..., and all (four) arrows in the diagram are predicates described by a formal system. L_1_DS, L_2_DS and T must be disjoints and t may use T. Note that these theories have the same language (Typol). The key-idea is to consider the formal system:

$$\mathcal{T} = T \cup L_1\_DS \cup L_2\_DS \cup t$$

We have (three or) four distinguished sets of rules in $\mathcal{T}$. Each sequent makes appeal to one particular set of rules, as it is in fact always the case in a Typol program: This is no more than a notion of modularity in our logic. Since these theories have no connection we could not have altered them in the join operation. We may only have added new facts... We shall say that the translation is correct iff [those

facts are wanted facts] two certain inference rules are valid in $\mathcal{T}$. The proof will use induction on the length of the proof.

Outline.

To provide an illustration of the method, we work on a specific example. Part 2 of the paper gives the dynamic semantics of Mini-ML (the purely applicative part of ML). Part 3 gives the dynamic semantics of CAM: "categorical abstract machine", a very interesting machine language developped by G. Cousineau and P.L. Curien [2]. Part 4 gives the translation of Mini-ML into CAM. We develop in Part 5 a notion of "approximate normal form" of a program (inspired by [18][9]), which will enable us to say that the criterion of correctness of translation given above deals with infinite programs as well. Part 6 introduces the method for proving the correctness of the translation. Finally part 7 gives the complete proof for our example: Mini-ML to CAM.

*

A complete proof of Mini-ML to CAM can be found in [11]. But this proof is done in a completely different context and style: M. Mauny proved that the CAM machine correctly simulates the β-reduction of the λ-calculus, by induction on the lenght of the transitions of the machine. Also W. Li [8] deals with correctness of translation, but he is interested in concurrent languages, and, taking the operational semantics 'a la Plotkin', considers equivalences of behaviours of transitions systems.

2. Dynamic semantics of Mini-ML

ML is a functional language with polymorphism and higher-order functions, first used in the proof system LCF [10]. Mini-ML consist in the purely applicative part of ML, more precisely a simple typed λ-calculus with constants, products, conditionals, and recursive function definitions. The abstract syntax of Mini-ML is given in Fig. 1. Simple programs in Mini-ML are for example, in *concrete* syntax, a term with both simultaneous definitions and block structure, or a simultaneous recursive definition:

$$\begin{array}{l}\textbf{let } (x,y) = (2,3)\\ \textbf{in let } (x,y) = (y,x) \textbf{ in } x\end{array}$$

$$\begin{array}{l}\textbf{letrec}(\mathit{even},\mathit{odd}) = (\lambda x.\, \textbf{if } x = 0 \textbf{ then } \mathit{true} \textbf{ else } \mathit{odd}(x-1),\\ \qquad\qquad \lambda x.\, \textbf{if } x = 0 \textbf{ then } \mathit{false} \textbf{ else } \mathit{even}(x-1))\\ \textbf{in } \mathit{even}\, 3\end{array}$$

2.1. The formal semantics of Mini-ML

Because of higher-order functions, the domain of semantic values of Mini-ML is slightly more complicated than for a less expressive language:

- integers $\mathbb{N}$
- truth values: *true*, *false*

sorts EXP, IDENT, PAT, NULLPAT

subsorts EXP$\supset$ NULLPAT, IDENT
PAT$\supset$ NULLPAT, IDENT

constructors

'Patterns'

pairpat	:	PAT×PAT	→	PAT
nullpat	:		→	NULLPAT

'Expressions'

ident	:		→	IDENT
number,false, true	:		→	EXP
apply, mlpair	:	EXP×EXP	→	EXP
lambda	:	PAT×EXP	→	EXP
let, letrec	:	PAT×EXP×EXP	→	EXP
if	:	EXP×EXP×EXP	→	EXP

Figure 1. Abstract Syntax of mini-ML

- closures: $[\![\lambda \text{P.E}, \rho]\!]$, where E is an expression and ρ is an environment. A closure is just a pair of a λ-expression and an environment.
- identifiers for predefined operators: *plus*, ...
- pairs of semantic values (which may in turn be pairs, so lists of semantic values maybe constructed)

Naturally the value of an expression e depends on the values of the identifiers that occur free in it. An *environment* ρ is an ordered list of pairs $P \mapsto \alpha$ where P is a pattern and α a value. Here is an example of environment: $x \mapsto 1 \cdot (x,y) \mapsto (\mathit{true}, 5)$.

We say that expression e evaluates to α in environment ρ if the theorem

$$\rho \vdash e : \alpha$$

can be derived from the formal system in Fig. 2.

2.2. Comments on the formal definition

In Figure 2, rules 1 to 3 associate values to integer or boolean literals. Rule i says that the evaluation of an expression begins with an initial environment (mapping a few predefined operators to "themselves"). Rule 4 constructs a closure for a λ-expression, pairing it with the environment. The value associated to an identifier must be looked up in the environment (rule 5). Given that the environment maps patterns to values, rather than identifiers to values, we need auxiliary rules, the set VAL_OF. Rules 6 and 7 associate values to conditional expression. Rule 8 is equally transparent.

The next rules deal with functional values. Rule 9 is a special case of rule 10, where E_1 evaluates to a predefined operator. Rule 10 is the general case of the evaluation of an application. Because of type-checking, the operator of an application can only evaluate to a functional value, i.e. a closure. This closure is taken apart, and its body is evaluated in its environment, prefixed with the parameter association P $\mapsto \alpha$. Note that the rule is valid whether P

program ML_DS is

use ML

$\rho, \rho_1 : PAT$;

$\alpha, \beta : VALUE$;

$$\frac{init\_env \vdash \mathrm{E} : \alpha}{\vdash \mathrm{E} : \alpha} \quad (i)$$

$$\rho \vdash \text{number}\,\mathrm{N} : \mathrm{N} \quad (1)$$

$$\rho \vdash \text{true} : true \quad (2)$$

$$\rho \vdash \text{false} : false \quad (3)$$

$$\rho \vdash \lambda \mathrm{P.E} : [\![\lambda \mathrm{P.E}, \rho]\!] \quad (4)$$

$$\frac{\rho \overset{val\_of}{\vdash} \text{ident}\,\mathrm{I} \mapsto \alpha}{\rho \vdash \text{ident}\,\mathrm{I} : \alpha} \quad (5)$$

$$\frac{\rho \vdash \mathrm{E}_1 : true \qquad \rho \vdash \mathrm{E}_2 : \alpha}{\rho \vdash \text{if}\,\mathrm{E}_1\,\text{then}\,\mathrm{E}_2\,\text{else}\,\mathrm{E}_3 : \alpha} \quad (6)$$

$$\frac{\rho \vdash \mathrm{E}_1 : false \qquad \rho \vdash \mathrm{E}_3 : \alpha}{\rho \vdash \text{if}\,\mathrm{E}_1\,\text{then}\,\mathrm{E}_2\,\text{else}\,\mathrm{E}_3 : \alpha} \quad (7)$$

$$\frac{\rho \vdash \mathrm{E}_1 : \alpha \qquad \rho \vdash \mathrm{E}_2 : \beta}{\rho \vdash (\mathrm{E}_1, \mathrm{E}_2) : (\alpha, \beta)} \quad (8)$$

$$\frac{\rho \vdash \mathrm{E}_1 : \text{ident}\,\mathrm{OP} \qquad \rho \vdash \mathrm{E}_2 : (\alpha, \beta) \qquad \overset{eval}{\vdash} \mathrm{OP}, \alpha, \beta : \gamma}{\rho \vdash \mathrm{E}_1\ \mathrm{E}_2 : \gamma} \quad (9)$$

$$\frac{\rho \vdash \mathrm{E}_1 : [\![\lambda \mathrm{P.E}, \rho_1]\!] \qquad \rho \vdash \mathrm{E}_2 : \alpha \qquad \mathrm{P} \mapsto \alpha \cdot \rho_1 \vdash \mathrm{E} : \beta}{\rho \vdash \mathrm{E}_1\ \mathrm{E}_2 : \beta} \quad (10)$$

$$\frac{\rho \vdash \mathrm{E}_2 : \alpha \qquad \mathrm{P} \mapsto \alpha \cdot \rho \vdash \mathrm{E}_1 : \beta}{\rho \vdash \text{let}\,\mathrm{P} = \mathrm{E}_2\,\text{in}\,\mathrm{E}_1 : \beta} \quad (11)$$

$$\frac{\rho_1 = \mathrm{P} \mapsto [\![\mathrm{E}_2, \rho_1]\!] \cdot \rho \qquad \rho_1 \vdash \mathrm{E}_1 : \alpha}{\rho \vdash \text{letrec}\,\mathrm{P} = \mathrm{E}_2\,\text{in}\,\mathrm{E}_1 : \alpha} \quad (12)$$

end ML_DS

Figure 2. The dynamic semantics of Mini-ML

is a pattern or a single variable. As, at evaluation time, $\lambda \mathrm{P.E_1\ E_2} = (\text{let}\,\mathrm{P} = \mathrm{E_2}\,\text{in}\,\mathrm{E_1})$, rule 11 appears to be an other optimisation of rule 10.[2]

The last rule, rule 12, defines in one and the same way the simple recursive functions and the mutually recursive ones. The environment in which $\mathrm{E_1}$ is evaluated is prefixed with a self-referencing closure. Notice that since $\rho \vdash \mathrm{E_2} : \alpha$ is a premisse of rule 10, we have an ML with call by value.

The separate set VAL_OF (see Fig. 3) defines rules to associate values to identifiers, given some environment. Since the environment maps patterns to values, the patterns must be traversed to find the relevant identifier. Furthermore, block structure is present in the environment because in rules 10 to 12 we have merely prefixed the environment with new associations.

set VAL_OF is

$$\text{ident}\,\mathrm{I} \mapsto \alpha \cdot \rho \vdash \text{ident}\,\mathrm{I} \mapsto \alpha \quad (1)$$

$$\frac{\rho \vdash \text{ident}\,\mathrm{I} \mapsto \alpha}{\text{ident}\,\mathrm{X} \mapsto \beta \cdot \rho \vdash \text{ident}\,\mathrm{I} \mapsto \alpha} \quad (\mathrm{X} \neq \mathrm{I}) \quad (2)$$

$$\frac{\mathrm{P_2} \mapsto \beta \cdot \mathrm{P_1} \mapsto \alpha \cdot \rho \vdash \text{ident}\,\mathrm{I} \mapsto \gamma}{(\mathrm{P_1}, \mathrm{P_2}) \mapsto (\alpha, \beta) \cdot \rho \vdash \text{ident}\,\mathrm{I} \mapsto \gamma} \quad (3)$$

$$\frac{\mathrm{P_2} \mapsto [\![\mathrm{E_2}, \rho_1]\!] \cdot \mathrm{P_1} \mapsto [\![\mathrm{E_1}, \rho_1]\!] \cdot \rho \vdash \text{ident}\,\mathrm{I} \mapsto \alpha}{(\mathrm{P_1}, \mathrm{P_2}) \mapsto [\![(\mathrm{E_1}, \mathrm{E_2}), \rho_1]\!] \cdot \rho \vdash \text{ident}\,\mathrm{I} \mapsto \alpha} \quad (4)$$

end VAL_OF

Figure 3. The ML environment rules

Rules 1 and 2 scan the environment until the first occurrence of an identifier is found, in a left to right scan. Rule 3 relies on the fact that, except for the case taken care of in rule 4, a pair of identifiers is bound to a value which is a pair. Hence searching is propagated to two new pattern-value pairs. Rule 4 takes care of the mutually recursive definitions. When a pair of patterns is associated to a single closure, this closure must come from a pair of functions. The environment of the closure is distributed over these functions and searching is propagated to simpler components. Thanks to this simple idea, the letrec rule 12 remains transparent, while accessing the environment is made only slightly more complex.

3. Dynamic semantics of CAM

The Categorical Abstract Machine [2] has its roots both in categories and in De Bruijn's notation for lambda-calculus. It is a very simple machine where, according to its inventors, "categorical terms can be considered as code acting on a graph of values". Instructions are few in number and quite close to real machine instructions. Instructions *car* and *cdr* serve in accessing data in the stack and the special instruction *rplac* is used to implement recursion. Predefined operations (such as addition, subtraction, division, etc.) may be added with the *op* instruction. The abstract syntax of CAM code is given in Fig. 4.

3.1. The formal semantics of CAM

The state of the CAM machine is a stack, whose top element may be viewed as a register. The values stored in this stack are:

- integers $\mathbb{N}$
- truth values: *true*, *false*
- closures of the form $[\![\mathrm{c}, \rho]\!]$, where c is a fragment of CAM code and ρ is a value, meant to denote an environment

[2] Optimisations of both dynamic semantics of Mini-ML and translation from Mini-ml to CAM are presented and prooved correct in [12].

sorts VALUE, COM, PROGRAM, COMS

subsorts COM⊃COMS

constructors

program	: COMS	→ PROGRAM
coms	: COM*	→ COMS
quote	: VALUE	→ COM
op,car,cdr,cons	:	→ COM
push,swap,app, rplac	:	→ COM
cur	: COMS	→ COM
branch	: COMS×COMS	→ COM
int,bool, null_value	:	→ VALUE

Figure 4. Abstract syntax of CAM code

- pairs of semantic values (which may in turn be pairs, so that trees may be constructed)

Except in the first rule, all sequents have the form

$$s \vdash c : s'$$

where c is CAM-code and s and s' are states of the CAM machine. The sequent $s \vdash c : s'$ may be read as *executing code c when the machine is in state s takes it to state s'*. The rules describing the transitions of the CAM appear in Fig. 5.

3.2. Comments on the formal definition

In Figure 5, rule 1 says that evaluating a program begins with an initial stack and ends with a value on top of the stack that is the result of the program. The initial stack contains closures corresponding to the predefined operators. Rule 2 and 3 deal with sequences of commands; rules 4 to 11 are self explanatory axioms. Rule 12 switches to an external evaluator EVAL for predefined operators.

Rule 13 and 14 define the *branch* instruction. It takes its (evaluated) condition from the top of the stack, and continues with either the true or the false part. The *cur* instruction is described in rule 15: $cur(\mathrm{C})$ builds a closure with the code C and the current environment (top of the stack) placing it on top of the stack. Rule 16 says that the *app* instruction must find on top of the stack a pair consisting of a closure and a parameter environment. Then the code of the closure is evaluated in a new environment: that of the closure prefixed by the parameter environment.

The last rule is the less intuitive one. An *rplac* instruction takes a pair consisting of an environment ρ and a variable V, followed by an environment ρ_1 on the stack. It identifies V and ρ_1 and places the pair (ρ, ρ_1) on the stack. Notice that each occurrence of V in ρ_1 has been replaced by ρ_1. The use of this instruction will be explained by the translation of the *letrec* instruction (see rule 9 on Fig. 6).

program CAM_DS is

use CAM

$s, s_1, s_2 : STACK$;

$\alpha, \beta : VALUE$;

$\rho, \rho_1 : ENV$;

$$\frac{init\_stack \vdash \mathrm{COMS} : \alpha}{\vdash program(\mathrm{COMS}) : \alpha} \quad (1)$$

$$s \vdash \emptyset : s \quad (2)$$

$$\frac{s \vdash \mathrm{COM} : s_1 \qquad s_1 \vdash \mathrm{COMS} : s_2}{s \vdash \mathrm{COM}; \mathrm{COMS} : s_2} \quad (3)$$

$$\alpha \cdot s \vdash quote(\mathrm{X}) : \mathrm{X} \cdot s \qquad (var(\mathrm{X})) \quad (4)$$

$$\alpha \cdot s \vdash quote(int\ \mathrm{N}) : \mathrm{N} \cdot s \quad (5)$$

$$\alpha \cdot s \vdash quote(bool\ \mathrm{T}) : \mathrm{T} \cdot s \quad (6)$$

$$(\alpha, \beta) \cdot s \vdash car : \alpha \cdot s \quad (7)$$

$$(\alpha, \beta) \cdot s \vdash cdr : \beta \cdot s \quad (8)$$

$$\alpha \cdot \beta \cdot s \vdash cons : (\beta, \alpha) \cdot s \quad (9)$$

$$\alpha \cdot s \vdash push : \alpha \cdot \alpha \cdot s \quad (10)$$

$$\alpha \cdot \beta \cdot s \vdash swap : \beta \cdot \alpha \cdot s \quad (11)$$

$$\frac{\overset{\mathrm{eval}}{\vdash}\ \mathrm{OP}, \alpha, \beta : \gamma}{(\alpha, \beta) \cdot \mathrm{S} \vdash op\ \mathrm{OP} : \gamma \cdot \mathrm{S}} \quad (12)$$

$$\frac{s \vdash \mathrm{C}_1 : s_1}{true \cdot s \vdash branch(\mathrm{C}_1, \mathrm{C}_2) : s_1} \quad (13)$$

$$\frac{s \vdash \mathrm{C}_2 : s_1}{false \cdot s \vdash branch(\mathrm{C}_1, \mathrm{C}_2) : s_1} \quad (14)$$

$$\rho \cdot s \vdash cur(\mathrm{C}) : [\![\mathrm{C}, \rho]\!] \cdot s \quad (15)$$

$$\frac{(\rho, \alpha) \cdot s \vdash \mathrm{C} : s_1}{([\![\mathrm{C}, \rho]\!], \alpha) \cdot s \vdash app : s_1} \quad (16)$$

$$\frac{\mathrm{V} = \rho_1}{(\rho, \mathrm{V}) \cdot \rho_1 \cdot s \vdash rplac : (\rho, \rho_1) \cdot s} \quad (17)$$

end CAM_DS

Figure 5. The definition of the CAM

4. Translation from Mini-ML to CAM

We are now ready to generate CAM code for mini-ML.

4.1. The formal system

The translation rules from mini-ML to CAM are given in Fig. 6. In these rules, except for rule 1, all sequents have the form:

$$\rho \vdash e \rightarrow c$$

where ρ is an environment, e is an ML_expression, and c is its translation into CAM-code. In words, the sequent may

be read as *in environment ρ, expression e is compiled into code c*. The notion of environment used in this translation is exactly the notion of an ML-pattern, i.e. a binary tree with identifiers at the leaves.

program ML_CAM is

use ML

use *CAM*

$c, c_1, c_2, c_3 : CAM;$

$\rho, \rho_1 : ENV;$

$$\frac{init\_pat \vdash \mathrm{E} \rightarrow c}{\vdash \mathrm{E} \rightarrow program(c)} \quad (1)$$

$$\rho \vdash \mathrm{number\,N} \rightarrow quote(int\ \mathrm{N}) \quad (2)$$

$$\rho \vdash \mathrm{true} \rightarrow quote(bool\ \text{"true"}) \quad (3)$$

$$\rho \vdash \mathrm{false} \rightarrow quote(bool\ \text{"false"}) \quad (4)$$

$$\frac{\rho \overset{\text{access}}{\vdash} \mathrm{ident\,I} \mapsto c}{\rho \vdash \mathrm{ident\,I} \rightarrow c} \quad (5)$$

$$\frac{\rho \vdash \mathrm{E_1} \rightarrow c_1 \qquad \rho \vdash \mathrm{E_2} \rightarrow c_2 \qquad \rho \vdash \mathrm{E_3} \rightarrow c_3}{\rho \vdash \mathrm{if\ E_1\ then\ E_2\ else\ E_3} \rightarrow push; c_1; branch(c_2, c_3)} \quad (6)$$

$$\frac{\rho \vdash \mathrm{E_1} \rightarrow c_1 \qquad \rho \vdash \mathrm{E_2} \rightarrow c_2}{\rho \vdash (\mathrm{E_1}, \mathrm{E_2}) \rightarrow push; c_1; swap; c_2; cons} \quad (7)$$

$$\frac{\rho \vdash \mathrm{E_1} \rightarrow c_1 \qquad (\rho, \mathrm{P}) \vdash \mathrm{E_2} \rightarrow c_2}{\rho \vdash \mathrm{let\ P = E_1\ in\ E_2} \rightarrow push; c_1; cons; c_2} \quad (8)$$

$$\frac{(\rho, \mathrm{P}) \vdash \mathrm{E_1} \rightarrow c_1 \qquad (\rho, \mathrm{P}) \vdash \mathrm{E_2} \rightarrow c_2}{\rho \vdash \mathrm{letrec\ P = E_1\ in\ E_2} \rightarrow \{push; quote(\rho_1); cons; push; c_1; swap; rplac; c_2\}} \quad (9)$$

$$\frac{(\rho, \mathrm{P}) \vdash \mathrm{E} \rightarrow c}{\rho \vdash \lambda \mathrm{P.E} \rightarrow cur(c)} \quad (10)$$

$$\frac{\rho \vdash \mathrm{E_2} \rightarrow c_2 \qquad \overset{\text{trans\_const}}{\vdash} \mathrm{E_1} \rightarrow c_1}{\rho \vdash \mathrm{E_1\ E_2} \rightarrow c_2; c_1} \quad (11)$$

$$\frac{\rho \vdash \mathrm{E_1} \rightarrow c_1 \qquad \rho \vdash \mathrm{E_2} \rightarrow c_2}{\rho \vdash \mathrm{E_1\ E_2} \rightarrow push; c_1; swap; c_2; cons; app} \quad (12)$$

end ML_CAM

Figure 6. Translation from mini-ML to CAM

4.2. Comments on the formal system

Translation of an ML program is invoked, in rule 1, with an initial environment *init_pat* that is merely a list of predefined functions. The environment builds up whenever one introduces new names (rules 9 and 10). It is consulted when one wants to generate code for an identifier (rule 5). Then an access path is computed in the ACCESS rule set (see Fig 7). The access path is a sequence of *car* and *cdr* instructions (a coding of the De Bruijn number associated to that occurrence of the identifier) that will access the corresponding value in the stack of the CAM.

set ACCESS is

$\varphi, \varphi_1 : ENV;$

$$\frac{\rho \mapsto \emptyset \vdash \mathrm{X} : c}{\rho \vdash \mathrm{X} : c} \quad (1)$$

$$\mathrm{ident\,X} \mapsto c \cdot \varphi \vdash \mathrm{ident\,X} : c \quad (2)$$

$$\frac{\varphi \vdash \mathrm{ident\,X} : c}{\mathrm{ident\,Y} \mapsto c' \cdot \varphi \vdash \mathrm{ident\,X} : c} \quad (\mathrm{Y} \neq \mathrm{X}) \quad (3)$$

$$\frac{\rho_2 \mapsto c; cdr \cdot \rho_1 \mapsto c; car \cdot \varphi \vdash \mathrm{X} : c'}{(\rho_1, \rho_2) \mapsto c \cdot \varphi \vdash \mathrm{X} : c'} \quad (4)$$

end ACCESS

Figure 7. Generating access paths for identifiers

Rules 2, 3, and 4 generate code for literal values. Rule 5 generates an access path for an identifier. Rules 6 and 7 are straightforward once the following inductive assertion is understood: *the code for an expression expects its evaluation environment on top of the stack, and it will overwrite this environment with its result.* Thus the environment must be saved, by a *push* instruction, when necessary.

Rule 8 shows how a run time environment is built up in the stack in parallel with the static environment. Rule 9 is a little surprising because it leaves a free variable ρ_1 in the code. This is a technique for leaving a reference to be resolved at run time. The instruction $quote(\rho_1)$ will leave (at execution time) a free variable on top of the stack. A closure will be built using the environment on top of the stack. Hence this closure will refer to variable ρ_1. Instruction *rplac* will tie a knot, freezing the value of ρ_1 as the appropriate closure. In this way, we build a self-referencing environment.

The remaining rules deal with closures. Rule 10 merely generates the instruction *cur* that constructs closures. Rule 11 concerns predefined operators. Finally, rule 12 is the general case for an application.

5. Infinite executions

Up to now, we only specify execution of programs that terminate. We develop here some material that enables us to deal with infinite programs as well, and present the method on Mini-ML. The first subsection present the general idea while the second subsection discuss the ML case in full.

5.1. General idea

In the spirit of denotational semantics, we add an undefined value $\perp$ to the semantic domain in Mini-ML, thus defining the set of approximate normal forms [18][9]for this language:

Definition 5.1. *An approximate normal form (a.n.f.) of Mini-ML is either*

- $\perp$,
- *a constant (integer, boolean, predefined identifier or closure), or*
- *a pair of a.n.f.*

and can only be obtained by this recursive definition.

Now, we add an axiom in our theory *ML_DS*, saying that the evaluation of an expression may return $\perp$:

$$\rho \vdash e : \perp$$

and we define the set of approximate normal forms an expression:

Definition 5.2. *α is an approximate normal form (a.n.f.) of t if*

- *α is an a.n.f.*
- *$\vdash t : \alpha$ holds*

Now, an infinite ML term as no semantic value - ML differs here from the λ-calculus - but an infinite set of a.n.f. For example, letrec $\mathrm{F} = \lambda \mathrm{X}.[\mathrm{X}.\mathrm{F}\ \mathrm{X}]$ in $\mathrm{F}\ 2$ as for a.n.f.: $\perp$ $[\perp.\perp]$ $[2.\perp]$ $[2.\perp.\perp]$ $[2.2.\perp]$ $\cdots$

5.2. Discussion on ML

Using the method developed in the previous subsection, our language ML become nondeterministic: for each expression we have the choice to evaluate it or not. But this is not truely nondeterminism as we shall now see.

Definition 5.3. *We define a partial order $\preceq$ on a.n.f. as follows: For all c_i constant, φ, φ_i, φ'_j a.n.f.:*

- $c_i \preceq c_j \Leftrightarrow i = j$
- $\perp \preceq \varphi$
- $(\varphi_i, \varphi_j) \preceq (\varphi'_i, \varphi'_j) \Leftrightarrow \varphi_i \preceq \varphi'_i \ \& \ \varphi_j \preceq \varphi'_j$

Fact. *$\Phi = (\{\text{a.n.f.}\}, \preceq)$ can be embedded in a cpo, by considering the set of directed sets of Φ, with the induced partial order, then identifying each a.n.f. α with the set of a.n.f. dominated by α.*

Definition 5.4. *From the partial order on a.n.f. we induce a partial order on environments, defined by extension:*

- $\emptyset \preceq \emptyset$
- $P \mapsto \alpha \cdot \rho \preceq P \mapsto \alpha' \cdot \rho' \Leftrightarrow \alpha \preceq \alpha' \ \& \ \rho \preceq \rho'$

Theorem 5.1. *For each ML term t, the set of a.n.f. of t is a directed set.*

Proof. Prove that for any $t, \rho_1, \rho_2, \rho, \alpha, \beta$, such that $\rho_1 \vdash t : \alpha$, $\rho_2 \vdash t : \beta$, $\rho_1 \preceq \rho$, $\rho_2 \preceq \rho$ there exists γ such that $\rho \vdash t : \gamma$, $\alpha \preceq \gamma$, $\beta \preceq \gamma$. The theorem follows, with $\rho_1 = \rho_2 = \rho = \emptyset$. The proof use induction on the length of the proof. We do not give it here, as it is very similar to the proof of the correctness of the translation, given in full later on.

□ Theorem 5.1

Fact. *In a cpo, a directed set admits a least upper bound.*

So we can define:

Definition 5.5. *For each term t, the limit of the set of a.n.f. of t, $\Phi(t)$, is its least upper bound.*

Now it is clear that, in adding the rule $\rho \vdash \mathrm{E} : \perp$, we have not really added non-determinism in our language, as we have a "Church-Rosser property":

$$\vdash p : \alpha \ \& \ \vdash p : \beta \ \Rightarrow \ \exists \gamma,\ \gamma \succeq \alpha,\ \gamma \succeq \beta \ s.t. \ \vdash p : \gamma$$

Furthermore, we have a limit of the sequence α_n such that $\vdash p : \alpha_n$. This guarantees the existence and unicity of the result, even if it is a limit.

6. Proof of translation

We are now ready to give our criteria of correctness of a translation. The first subsection present those (two) criteria, the second one shows that they are adequate criteria of correctness of a translation, while the last subsection shows that they are equivalent in simple cases.

6.1. General case

We follow here the idea developped in the introduction. We work in a theory

$$\mathcal{T} = T \cup L_1\_DS \cup L_2\_DS \cup t$$

and our criteria of correctness of translation are as follows:

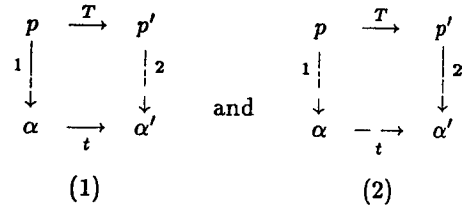
Definition 6.1. *The translation is correct iff the following inference rules are valid in $\mathcal{T}$:*

$$\frac{\overset{1}{\vdash} p : \alpha \qquad \overset{T}{\vdash} p \to p' \qquad \overset{t}{\vdash} \alpha \to \alpha'}{\overset{2}{\vdash} p' : \alpha'} \tag{1}$$

$$\exists \alpha \qquad \frac{\overset{T}{\vdash} p \to p' \qquad \overset{2}{\vdash} p' : \alpha'}{\overset{1}{\vdash} p : \alpha \quad \& \quad \overset{t}{\vdash} \alpha \to \alpha'} \tag{2}$$

where p and p' are source and object programs, α and α' are semantic values (or approximate normal forms), and the superscripts of the turnstiles denote the set of rules under consideration.[3]

These inference rules are expressing commutativity of a diagram. In pictures we ask for:



(1) completeness (2) soundness

[3] All implicit quantifiers are universal quantifiers.

where "$\longrightarrow$" denote the given facts and "$- \rightarrow$" denote the facts to prove.

The second inference rule is somewhat unusual. However, notice that an instance of this rule is not too surprising as a sub-proof-tree. Anyway, a more usual form of this rule would be:

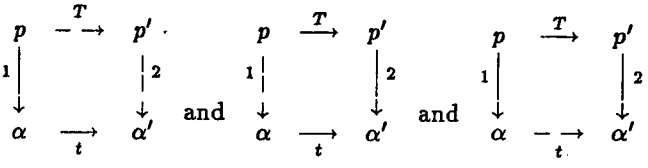
$$\exists \alpha \quad \frac{\overset{T}{\vdash} p \rightarrow p' \qquad \overset{2}{\vdash} p' : \alpha'}{\underset{1 \cup t}{\vdash} p : \alpha \;\&\; \alpha \rightarrow \alpha'} \qquad (2a)$$

This form is equivalent to the previous one for the theory 1 and t have no connection. But experience shows that Rule (2) is easier to manipulate, in our context, than Rule (2a).

Note that our criteria of correctness are sufficient for all programs, no-matter whether they terminate or not, are erroneus or not: α and α' are approximate normal forms of p and p', or errors.

6.2. Discussion

For the sake of completeness, we shall examine all other possible diagrams which would be chosen as criteria of correctness. We take as hypothesis that T and t are always defined, and there exists an a.n.f. of any source term (which means that all typed-checked programs are executable). We do not consider diagrams which are obviously too strong requirements for the correctness of a translation. The remaining diagrams, apart from (1) and (2), are mainly:[4]



(1′) (2′) (3)

completeness soundness correctness

It is easy to check that T is defined on all program and (1) implies (1'). Also, (2') is too strong in general, and is equivalent to (2) in the case where t is a one-one mapping. Now (3) is too strong in the case where L_1 and L_2 are nondeterministic. It appears that (1) and (2) are sufficient criteria of correctness fo nondeterminism and enables one to avoid the difficult problem of equality of semantic domains [16]. Thus, (1) and (2) seems to be adequate criteria of correctness of a translation.

6.3. Simple (but not infrequent) cases

In simple cases, rule (1) and rule (2) are equivalent, since it is easy to prove the following

[4] four other diagrams of minor interest are implied by (1) and the hypothesis on the formal systems.

Theorem 6.1. *If* $\overset{1}{\vdash}$ *and* $\overset{2}{\vdash}$ *are "deterministic" (i.e.* $\vdash p:\alpha \;\&\; \vdash p:\beta \;\Rightarrow\; \alpha = \beta$*) and if t is a one-one mapping (i.e.* $\forall\alpha$ *(resp.* β*)* $\exists!\beta$ *(resp.* α*) s.t.* $\overset{t}{\vdash} \alpha \rightarrow \beta$*) then the inference rules (1) and (2) are equivalent.*

6.4. Induction on the length of the proof

To prove the validity of an inference rule, we have to prove that for each proof of the premisses, we can exibit a proof of the conclusion. For that we shall use *induction on the length of the proofs* of the premisses. We are allowed to use induction on the length of the proof for we are working in the "deductive system", and/or we only consider "syntactic models". Let's say we want to do semantics and still be "purely syntactic"... Proofs are for us a -very- formal-game of "dominos". We could also attempt to use structural induction on the source program, but, as we shall see in our example, this induction is not sufficient to carry out the proof.

We are ready now to prove the correctness of the translation from Mini-ML to CAM. Unfortunately, in this case t is not one-one (because of the closures), so we have to prove that both Rule (1) and Rule (2) hold.

7. Proof of the translation from Mini-ML to CAM

We have already described three of the four inference systems required: ml_ds, ml_cam and cam_ds. They specify respectively the dynamic semantics of Mini-ML, the translation from Mini-ML to CAM and the dynamic semantics of CAM. Now we must give the formal system, t, describing the translation of semantic values.

7.1. Translation of semantic values

We need here two auxiliary definitions. Given ρ, an environment used by ML, consisting of a list of mapping of the form $[\mathrm{P} \mapsto \alpha, \mathrm{Q} \mapsto \beta, \mathrm{R} \mapsto \gamma]$, we define:

Definition 7.1. *$\bar{\rho}$ is an environment used by the translation. It is the "pattern" corresponding to ρ and defined by the rules:*

- $\overline{\varnothing} = \_$
- $\overline{\mathrm{P} \mapsto \alpha} = \mathrm{P}$
- $\overline{\rho \cdot \rho_1} = (\bar{\rho}_1, \bar{\rho})$

Definition 7.2. *$\vec{\rho}$ is a value put on the stack used by CAM. It is the "term" corresponding to ρ and defined by:*

- $\vec{\varnothing} = 0$
- $\overline{\mathrm{P} \mapsto \alpha} = t(\alpha)$
- $\overline{\rho \cdot \rho_1} = (\vec{\rho}_1, \vec{\rho})$

Example. For $\rho = [\mathrm{P} \mapsto 1, \mathrm{Q} \mapsto 2, \mathrm{R} \mapsto 3]$ we have $\bar{\rho} = (((\_, \mathrm{R}), \mathrm{Q}), \mathrm{P})$ and $\vec{\rho} = (((0,3),2),1)$.

The defioition of t on semantic values is given in Fig. 8. It is quite natural, with the possible exception of closures.

$$\vdash t(\mathrm{N}) = \mathrm{N} \quad (1)$$

$$\vdash t(\mathrm{T}) = \mathrm{T} \quad (2)$$

$$\vdash t((\alpha,\beta)) = (t(\alpha), t(\beta)) \quad (3)$$

$$\frac{\overset{\text{trans\_const}}{\vdash \quad \mathrm{OP} \to \mathrm{OP}'}}{\vdash t(\text{ident OP}) = [\![cdr; \mathrm{OP}', 0]\!]} \quad (4)$$

$$\frac{\tilde{\rho} \overset{\text{ml\_cam}}{\vdash} \lambda \mathrm{P.E} \to cur(c)}{\vdash t([\![\lambda \mathrm{P.E}, \rho]\!]) = [\![c, \vec{\rho}]\!]} \quad (5)$$

$$\vdash t([\![(\alpha,\beta),\rho]\!]) = (t([\![\alpha,\rho]\!]), t([\![\beta,\rho]\!])) \quad (6)$$

$$\vdash t(\bot) = \bot \quad (7)$$

Figure 8. The definition of t on semantic values

We have presented t as a function in the interest of compactness in the layout of proof trees. Rules (1) and (2) say that the translation on simple semantic values is the identity. Rule (3) says that the translation of a pair is the pair of the translations. The translation of a predefined operator (plus,etc...) is -in short- a closure of the code corresponding to this operator (Rule (4)). The translation of a closure of a λ-expression is the corresponding closure(Rule (5)). The translation of a closure of a pair of expressions is the pair of the translation of the closure of each expression (Rule (6)). Finally t is the identity on $\bot$ (Rule (7)). This rule is for the case of partial evaluation of the program.

The only -little- difficulty is for closures: in this case, $t(\alpha)$ is defined in term of $\vec{\rho}$ and $\vec{\rho}$ is defined in term of $t(\alpha)$. In fact, there is no mistery: the translation of a graph is a graph. For example, consider $\rho = \mathrm{P} \mapsto [\![\mathrm{E}, \rho]\!]$. We have $\tilde{\rho} = \mathrm{P}$ and $\dfrac{[\mathrm{P}] \overset{\text{ml\_cam}}{\vdash} \mathrm{E} \to cur(c)}{\vec{\rho} = t([\![\mathrm{E},\rho]\!]) = [\![c, \vec{\rho}]\!]}$. So $\alpha = [\![\mathrm{E},\rho]\!] = [\![\mathrm{E}, \mathrm{P} \mapsto \alpha]\!]$ is a graph and $t(\alpha) = [\![c, t(\alpha)]\!]$ is also a graph.

We are now ready to proceed with the complete proof.

7.2. Proof of rule (1)

We have to prove that the following inference rule:

$$\frac{\overset{\text{ml\_ds}}{\vdash \quad e : \alpha} \qquad \overset{\text{ml\_cam}}{\vdash \quad e \to c}}{\overset{\text{cam\_ds}}{\vdash \quad c : t(\alpha)}}$$

is valid in the theory $\mathcal{T} = \text{ml\_ds} \cup \text{ml\_cam} \cup \text{cam\_ds} \cup \text{t}$.

For each proof tree for the premisses we must exhibit a proof tree for the conclusion. Let's make one step in this direction. We must construct the following proof tree:

$$\frac{\dfrac{\overset{\text{ml\_ds}}{init\_env \vdash e : \alpha}}{\overset{\text{ml\_ds}}{\vdash e : \alpha}} \qquad \dfrac{\overset{\text{ml\_cam}}{init\_pat \vdash e \to c}}{\overset{\text{ml\_cam}}{\vdash, \quad e \to program(c)}}}{\dfrac{\overset{\text{cam\_ds}}{init\_stack \vdash c : t(\alpha)}}{\overset{\text{cam\_ds}}{\vdash program(c) : t(\alpha)}}}$$

This suggests what inference rule we should attempt to prove. It is stronger than the above inference rule, as it is often the case in proofs by induction. By definition, $\overline{init\_env} = init\_pat$ and $\overline{init\_env} = init\_stack$. So we shall prove that, for all expressions e of Mini-ML, for all environment ρ, and for all stack S of CAM:

$$\frac{\rho \overset{\text{ml\_ds}}{\vdash} e : \alpha \qquad \bar{\rho} \overset{\text{ml\_cam}}{\vdash} e \to c}{\vec{\rho} \cdot \mathrm{S} \overset{\text{cam\_ds}}{\vdash} c : t(\alpha) \cdot \mathrm{S}}$$

is valid, by induction on the length of the proofs of the premisses.

For each step of the induction, we shall draw a proof tree containing the three proof trees under consideration. The symbol $\vdash$ will be overloaded as the set of rules involved will become evident from the context. Uses of lemma or hypothesis of induction will be indicated by (lemma L) or (induction).

We have to consider all possible proof trees of $\rho \overset{\text{ml\_ds}}{\vdash} e : \alpha$ and $\bar{\rho} \overset{\text{ml\_cam}}{\vdash} e \to c$. For simple cases, when there is one inference rule in ml_ds and one in ml_cam (with the exception of the $\bot$ - rule) dealing with a given constructor of ML, this looks like structural induction. The most interesting case, for which structural induction on the source term is not powerfull enough, is the general case of an application (Rules ml_ds.10 & ml_cam.13).

Rules ml_ds.1 & ml_cam.2: $e = \text{number N}$.

$$\frac{\rho \vdash \text{number N} : \mathrm{N} \qquad \bar{\rho} \vdash \text{number N} \to quote(int\ \mathrm{N})}{\vec{\rho} \cdot \mathrm{S} \vdash quote(int\ \mathrm{N}) : \mathrm{N} \cdot \mathrm{S}}$$

For this we have only used rules ml_ds.2 & ml_cam.2 and rules cam_ds.5. No induction was needed as the proof trees are of length 1.

$\bot$-Rules in ml_ds & ml_cam: $e = \text{number N}$.

$$\frac{\rho \vdash \text{number N} : \bot \qquad \bar{\rho} \vdash \text{number N} \to quote(int\ \mathrm{N})}{\vec{\rho} \cdot \mathrm{S} \vdash quote(int\ \mathrm{N}) : \bot \cdot \mathrm{S}}$$

For each proof tree of $\rho \vdash e : \alpha$ we can use the axiom $\rho \vdash e : \bot$. Each time we have made that choice in ml_ds we can make the similar choice in cam_ds and use the axiom $\alpha \cdot \mathrm{S} \vdash c : \bot \cdot \mathrm{S}$. So each one of these cases will be as trivial as the proof above, and we will not consider them in the following.

Rules ml_ds.2, .3 & ml_cam.3, .4: The proofs are very similar to the previous one.

Rules ml_ds.4 & ml_cam.10: $e = \lambda \text{P.E}$.

$$\frac{\rho \vdash \lambda \text{P.E} : [\![\lambda \text{P.E}, \rho]\!] \qquad \frac{(\bar{\rho}, \text{P}) \vdash \text{E} \rightarrow c}{\bar{\rho} \vdash \lambda \text{P.E} \rightarrow cur(c)}}{\vec{\rho} \cdot \text{s} \vdash cur(c) : [\![c, \bar{\rho}]\!] \cdot \text{s}}$$

Rules ml_ds.5 & ml_cam.5: $e = \text{ident I}$.

$$\frac{\dfrac{\rho \overset{\text{val\_of}}{\vdash} \text{ident I} \mapsto \alpha}{\rho \vdash \text{ident I} : \alpha} \qquad \dfrac{\bar{\rho} \overset{\text{access}}{\vdash} \text{ident I} \mapsto c}{\bar{\rho} \vdash \text{ident I} \rightarrow c}}{\vec{\rho} \cdot \text{s} \vdash c : t(\alpha) \cdot \text{s} \qquad \text{(lemma "environment simulation")}}$$

Note: This proof tree is unusual in that hypothesis used for derive the conclusion are not written just above it: we have not rewritten these two hypothesis, as one usually do. This will be general in the paper: in order to make our proof trees managable, we shall not rewrite the hypothesis at each stage.

Now, we have used a lemma which says that the ML environment is correctly simulated in the Cam:

Lemma "environment simulation".

$$\frac{\rho \overset{\text{val\_of}}{\vdash} \text{ident I} \mapsto \alpha \qquad \bar{\rho} \overset{\text{access}}{\vdash} \text{ident I} \mapsto c}{\vec{\rho} \cdot \text{s} \vdash c : t(\alpha) \cdot \text{s}}$$

Proof. It is easy to prove, by induction on the length of the proof, the following stronger rule:

$$\frac{\rho \overset{\text{val\_of}}{\vdash} \text{ident I} \mapsto \alpha \qquad \varphi_{c'}(\bar{\rho}) \overset{\text{access}}{\vdash} \text{ident I} \mapsto c'; c}{\vec{\rho} \cdot \text{s} \vdash c : t(\alpha) \cdot \text{s}}$$

where $\varphi_c(\rho)$ is the environment mapping the identifiers of ρ to their access paths in ρ, prefixed by c. The definition of φ_c is as follows:

- $\varphi_c(\emptyset) = \emptyset$
- $\varphi_c(\rho) = \rho \mapsto c$
- $\varphi_c(\rho, \rho_1) = \varphi_{c;cdr}(\rho_1) \cdot \varphi_{c;car}(\rho)$

For example, $\varphi_c[(((\text{-}, \text{R}), \text{Q}), \text{P})] = \text{P} \mapsto c; cdr \cdot \text{Q} \mapsto c; car; cdr \cdot \text{R} \mapsto c; car; car$.

□ *Lemma "environment simulation"*

From now on we shall not give the proof tree in cam_ds in full details. Executions of sequences of commands, or *push*, *car*... will be skipped.

Rules ml_ds.7 & ml_cam.6: $e = \text{if E}_1 \text{ then E}_2 \text{ else E}_3$. The "false case" is similar to the previous (true) case.

Rules ml_ds.8 & ml_cam.7: $e = (\text{E}_1, \text{E}_2)$.

$$\frac{\dfrac{\rho \vdash \text{E}_1 : \alpha \quad \rho \vdash \text{E}_2 : \beta}{\rho \vdash (\text{E}_1, \text{E}_2) : (\alpha, \beta)} \qquad \dfrac{\bar{\rho} \vdash \text{E}_1 \rightarrow c_1 \quad \bar{\rho} \vdash \text{E}_2 \rightarrow c_2}{\bar{\rho} \vdash (\text{E}_1, \text{E}_2) \rightarrow push; c_1; swap; c_2; cons}}{\dfrac{\dfrac{\vec{\rho} \cdot \vec{\rho} \cdot \text{s} \vdash c_1 : t(\alpha) \cdot \vec{\rho} \cdot \text{s} \text{ (induction)}}{\vec{\rho} \cdot t(\alpha) \cdot \text{s} \vdash c_2 : t(\beta) \cdot t(\alpha) \cdot \text{s} \text{ (induction)}}}{\vec{\rho} \cdot \text{s} \vdash push; c_1; swap; c_2; cons : (t(\alpha), t(\beta)) \cdot \text{s}}}$$

For Rule ml_ds.9, we use a Lemma on eval, which is taken as hypothesis:

Lemma "$eval_1$".

$$\frac{\overset{\text{eval}}{\vdash} \text{OP}, \alpha, \beta : \gamma \qquad \overset{\text{trans\_const}}{\vdash} \text{ident OP} \rightarrow c}{\overset{\text{eval}}{\vdash} c, t(\alpha), t(\beta) : t(\gamma)}$$

$$\frac{\dfrac{\rho \vdash \text{E}_1 : \text{"true"} \qquad \rho \vdash \text{E}_2 : \alpha}{\rho \vdash \text{if E}_1 \text{ then E}_2 \text{ else E}_3 : \alpha} \qquad \dfrac{\bar{\rho} \vdash \text{E}_1 \rightarrow c_1 \qquad \bar{\rho} \vdash \text{E}_2 \rightarrow c_2 \qquad \bar{\rho} \vdash \text{E}_3 \rightarrow c_3}{\bar{\rho} \vdash \text{if E}_1 \text{ then E}_2 \text{ else E}_3 \rightarrow push; c_1; branch(c_2, c_3)}}{\dfrac{\vec{\rho} \cdot \vec{\rho} \cdot \text{s} \vdash c_1 : \text{"true"} \cdot \vec{\rho} \cdot \text{s} \text{ (induction)} \qquad \dfrac{\vec{\rho} \cdot \text{s} \vdash c_2 : t(\alpha) \cdot \text{s} \text{ (induction)}}{\text{"true"} \cdot \vec{\rho} \cdot \text{s} \vdash branch(c_2, c_3) : t(\alpha) \cdot \text{s}}}{\vec{\rho} \cdot \text{s} \vdash push; c_1; branch(c_2, c_3) : t(\alpha) \cdot \text{s}}}$$

Figure 9. **Rules ml_ds.6 & ml_cam.6:** $e = \text{if E}_1 \text{ then E}_2 \text{ else E}_3$.

$$\frac{\dfrac{\rho \vdash \text{E}_1 : \text{ident OP} \quad \rho \vdash \text{E}_2 : (\alpha, \beta) \qquad \overset{\text{eval}}{\vdash} \text{OP}, \alpha, \beta : \gamma}{\rho \vdash \text{E}_1 \text{ E}_2 : \gamma} \qquad \dfrac{\bar{\rho} \vdash \text{E}_2 \rightarrow c_2 \qquad \overset{\text{trans\_const}}{\vdash} \text{E}_1 \rightarrow c_1}{\bar{\rho} \vdash \text{E}_1 \text{ E}_2 \rightarrow c_2; c_1}}{\dfrac{\vec{\rho} \cdot \text{s} \vdash c_2 : (t(\alpha), t(\beta)) \cdot \text{s} \text{ (induction)} \qquad \dfrac{\overset{\text{eval}}{\vdash} c_1, t(\alpha), t(\beta) : t(\gamma) \text{ (lemma "}eval_1\text{")}}{(t(\alpha), t(\beta)) \cdot \text{s} \vdash c_1 : t(\gamma) \cdot \text{s}}}{\vec{\rho} \cdot \text{s} \vdash c_2; c_1 : t(\gamma) \cdot \text{s}}}$$

Figure 10. **Rules ml_ds.9 & ml_cam.11:** $e = \text{E}_1 \text{ E}_2$ with $\text{E}_1 = \text{ident OP}$.

$$\dfrac{\dfrac{\rho \vdash \mathrm{E}_1 : \mathrm{ident\,OP} \quad \rho \vdash \mathrm{E}_2 : (\alpha,\beta) \quad \overset{\text{eval}}{\vdash} \mathrm{OP}, \alpha, \beta : \gamma}{\rho \vdash \mathrm{E}_1\ \mathrm{E}_2 : \gamma} \qquad \dfrac{\bar{\rho} \vdash \mathrm{E}_1 \to c_1 \qquad \bar{\rho} \vdash \mathrm{E}_2 \to c_2}{\bar{\rho} \vdash \mathrm{E}_1\ \mathrm{E}_2 \to push; c_1; swap; c_2; cons; app}}{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{\text{eval}}{\vdash} t(op), t(\alpha), t(\beta) : t(\gamma)\ \text{(lemma "eval")}}{(t(\alpha), t(\beta)) \cdot \mathrm{s} \vdash t(op) : t(\gamma) \cdot \mathrm{s}}}{(0, (t(\alpha), t(\beta))) \cdot \mathrm{s} \vdash cdr; t(op) : t(\gamma) \cdot \mathrm{s}}}{([\![cdr; t(op), 0]\!], (t(\alpha), t(\beta))) \cdot \mathrm{s} \vdash app : t(\gamma) \cdot \mathrm{s}}}{\vec{\rho} \cdot [\![cdr; t(op), 0]\!] \cdot \mathrm{s} \vdash c_2 : (t(\alpha), t(\beta)) \cdot [\![cdr; t(op), 0]\!] \cdot \mathrm{s}\ \text{(induction)}}}{\vec{\rho} \cdot \vec{\rho} \cdot \mathrm{s} \vdash c_1 : [\![cdr; t(op), 0]\!] \cdot \vec{\rho} \cdot \mathrm{s}\ \text{(induction)}}}{\vec{\rho} \cdot \mathrm{s} \vdash push; c_1; swap; c_2; cons; app : t(\gamma) \cdot \mathrm{s}}$$

$\overset{\text{trans\_const}}{\vdash}\ \mathrm{ident\,OP} \to t(op)$

Figure 11. **Rules ml_ds.9 & ml_cam.13:** $e = \mathrm{E}_1\ \mathrm{E}_2$ with E_1 executes to ident OP.

$$\dfrac{\dfrac{\rho \vdash \mathrm{E}_2 : \alpha \quad \rho \vdash \mathrm{E}_1 : [\![\lambda \mathrm{P}.\mathrm{E}, \rho_1]\!] \quad \mathrm{P} \mapsto \alpha \cdot \rho_1 \vdash \mathrm{E} : \beta}{\rho \vdash \mathrm{E}_1\ \mathrm{E}_2 : \beta} \qquad \dfrac{(\bar{\rho}_1, \mathrm{P}) \vdash \mathrm{E} \to c \qquad \bar{\rho} \vdash \mathrm{E}_1 \to c_1 \qquad \bar{\rho} \vdash \mathrm{E}_2 \to c_2}{\bar{\rho} \vdash \mathrm{E}_1\ \mathrm{E}_2 \to push; c_1; swap; c_2; cons; app}}{\dfrac{\dfrac{\dfrac{\dfrac{(\vec{\rho}_1, t(\alpha)) \cdot \mathrm{s} \vdash c : t(\beta) \cdot \mathrm{s}\ \text{(induction)}}{([\![c, \vec{\rho}_1]\!], t(\alpha)) \cdot \mathrm{s} \vdash app : t(\beta) \cdot \mathrm{s}}}{\vec{\rho} \cdot [\![c, \vec{\rho}_1]\!] \cdot \mathrm{s} \vdash c_2 : t(\alpha) \cdot [\![c, \vec{\rho}_1]\!] \cdot \mathrm{s}\ \text{(induction)}}}{\vec{\rho} \cdot \vec{\rho} \cdot \mathrm{s} \vdash c_1 : [\![c, \vec{\rho}_1]\!] \cdot \vec{\rho} \cdot \mathrm{s}\ \text{(induction)}}}{\vec{\rho} \cdot \mathrm{s} \vdash push; c_1; swap; c_2; cons; app : t(\beta) \cdot \mathrm{s}}$$

Figure 12. **Rules ml_ds.10 & ml_cam.13:** $e = \mathrm{E}_1\ \mathrm{E}_2$, general case.

$$\dfrac{\dfrac{\rho \vdash \mathrm{E}_1 : \alpha \qquad \mathrm{P} \mapsto \alpha \cdot \rho \vdash \mathrm{E}_2 : \beta}{\rho \vdash \mathrm{let\ P} = \mathrm{E}_1\ \mathrm{in}\ \mathrm{E}_2 : \beta} \qquad \dfrac{\bar{\rho} \vdash \mathrm{E}_1 \to c_1 \qquad (\bar{\rho}, \mathrm{P}) \vdash \mathrm{E}_2 \to c_2}{\bar{\rho} \vdash \mathrm{let\ P} = \mathrm{E}_1\ \mathrm{in}\ \mathrm{E}_2 \to push; c_1; cons; c_2}}{\dfrac{\vec{\rho} \cdot \vec{\rho} \cdot \mathrm{s} \vdash c_1 : t(\alpha) \cdot \vec{\rho} \cdot \mathrm{s}\ \text{(induction)} \qquad (\vec{\rho}, t(\alpha)) \cdot \mathrm{s} \vdash c_2 : t(\beta) \cdot \mathrm{s}\ \text{(induction)}}{\vec{\rho} \cdot \mathrm{s} \vdash push; c_1; cons; c_2 : t(\beta) \cdot \mathrm{s}}$$

Figure 13. **Rules ml_ds.11 & ml_cam.8:** $e = \mathrm{let\ P} = \mathrm{E}_1\ \mathrm{in}\ \mathrm{E}_2$.

$$\dfrac{\dfrac{\rho_1 = \mathrm{P} \mapsto [\![\mathrm{E}_1, \rho_1]\!] \cdot \rho \quad \rho_1 \vdash \mathrm{E}_2 : \beta}{\rho \vdash \mathrm{letrec\ P} = \mathrm{E}_1\ \mathrm{in}\ \mathrm{E}_2 : \beta} \qquad \dfrac{\dfrac{\rho_1 \vdash \mathrm{E}_1 : v_1}{(\bar{\rho}, \mathrm{P}) \vdash \mathrm{E}_1 \to c} \qquad (\bar{\rho}, \mathrm{P}) \vdash \mathrm{E}_2 \to c'}{\bar{\rho} \vdash \mathrm{letrec\ P} = \mathrm{E}_1\ \mathrm{in}\ \mathrm{E}_2 \to push; quote(\mathrm{X}); cons; push; c; swap; rplac; c}}{\dfrac{\dfrac{(\vec{\rho}, \mathrm{X}) \cdot (\vec{\rho}, \mathrm{X}) \cdot \mathrm{s} \vdash c : t(v_1) \cdot (\vec{\rho}, \mathrm{X}) \cdot \mathrm{s}\ *1 \qquad \dfrac{\mathrm{X} = t(v_1) \qquad (\vec{\rho}, t(v_1)) \cdot \mathrm{s} \vdash c' : t(\beta) \cdot \mathrm{s}\ *2}{(\vec{\rho}, \mathrm{X}) \cdot t(v_1) \cdot \mathrm{s} \vdash rplac : (\vec{\rho}, t(v_1)) \cdot \mathrm{s}}}{\vec{\rho} \cdot \mathrm{s} \vdash push; quote(\mathrm{X}); cons; push; c; swap; rplac; c' : t(\beta) \cdot \mathrm{s}}$$

Figure 14. **Rules ml_ds.12 & ml_cam.9:** $e = \mathrm{letrec\ P} = \mathrm{E}_1\ \mathrm{in}\ \mathrm{E}_2$.

Rules ml_ds.12 & ml_cam.9: $e = \text{letrec } \mathrm{P} = \mathrm{E}_1 \text{ in } \mathrm{E}_2$.

The proof tree is given in Fig. 14. To draw it, we have used Theorem 5.1, which state that $\Phi(\mathrm{E}_1)$ is not empty, so $\forall \mathrm{E}_1$, $\exists v_1$ s.t. $\rho_1 \vdash \mathrm{E}_1 : v_1$. Uses of hypothesis of induction were valid with some extra hypothesis: The first one (*1) needs $\mathrm{x} = t([\![\mathrm{E}_1, \rho_1]\!])$, while the second one (*2) needs $t(v_1) = t([\![\mathrm{E}_1, \rho_1]\!])$. These two extra hypothesis are valid by the following lemma:

Lemma λ_1. $\rho \vdash e : v, e$ is a list of λ-exp. $\Rightarrow t(v) = t([\![e, \rho]\!])$

Proof. The proof uses induction on the length of the proof of $\rho \vdash e : v$.

1st case: $e = \lambda \mathrm{P.E}$. We have $\rho \vdash e : [\![\lambda \mathrm{P.E}, \rho]\!]$. So $t(v) = t([\![\lambda \mathrm{P.E}, \rho]\!]) = t([\![e, \rho]\!])$.

2nd case: $e = (\alpha, \beta)$, $\alpha, \beta \in \lambda$-exp.

$$\frac{\rho \vdash \alpha : \alpha' \qquad \rho \vdash \beta : \beta'}{\rho \vdash (\alpha, \beta) : (\alpha', \beta')}$$

$$\begin{aligned} t((\alpha', \beta')) &= (t(\alpha'), t(\beta')) \text{ by definition of t} \\ &= (t([\![\alpha, \rho]\!]), t([\![\beta, \rho]\!])) \text{ by hyp. of induction} \\ &= t([\![(\alpha, \beta), \rho]\!]) \text{ by definition of t} \end{aligned}$$

□ *Lemma* λ_1

□ Rule (1)

7.3. Proof of rule (2)

The proof of the second inference rule will complete the proof of the correctness of our translation.

$$\exists \alpha \qquad \frac{\overset{\text{ml\_cam}}{\vdash \quad e \to c} \qquad \overset{\text{cam\_ds}}{\vdash \quad c : \alpha'}}{\overset{\text{ml\_ds}}{\vdash \quad e : \alpha} \qquad \overset{\text{t}}{\vdash \alpha \to \alpha'}}$$

As before, we shall prove a stronger rule:

$$\exists \alpha, \rho \qquad \frac{\overset{\text{ml\_cam}}{\bar{\rho} \quad \vdash \quad e \to c} \qquad \overset{\text{cam\_ds}}{\vec{\rho} \cdot \mathrm{s} \quad \vdash \quad c : \alpha' \cdot \mathrm{s}}}{\overset{\text{ml\_ds}}{\rho \quad \vdash \quad e : \alpha} \qquad \overset{\text{t}}{\vdash \alpha \to \alpha'}}$$

Here again, we can give an equivalent form of this rule, for the theory ml_ds and t have no connection and for the theorem of deduction holds:

$$\exists \alpha, \rho \qquad \frac{\overset{\text{ml\_cam}}{\bar{\rho} \quad \vdash \quad e \to c} \qquad \overset{\text{cam\_ds}}{\vec{\rho} \cdot \mathrm{s} \quad \vdash \quad c : \alpha' \cdot \mathrm{s}}}{\overset{\text{ml\_dsUt}}{\vdash \quad \rho \Rightarrow e : \alpha \ \& \ \alpha \to \alpha'}}$$

The proof will again use induction on the length on the proof. Proof of rule (2) is mostly similar to proof of rule (1),we shall not give it in full detail. The differences concern the apply and letrec rules, and the constant use of the following lemma:

Lemma Cam. Every Cam code translating an ML-expression preserves the stack. More formaly:

$$\exists \beta \qquad \frac{\overset{\text{l\_cam}}{\rho \quad \vdash \quad e \to c} \qquad \overset{\text{cam\_ds}}{\alpha \cdot s \quad \vdash \quad c : s'}}{\overset{\text{cam\_ds}}{\alpha \cdot s \quad \vdash \quad c : \beta \cdot s}}$$

Proof. The proof is obvious, by induction on the length of the proof, exept maybe for rule ml_cam.12 (see Fig. 15).

□ *Lemma Cam*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\rho' \vdash \mathrm{E}_3 \to c_3}{\rho \vdash \mathrm{E}_1 \to c_1 \qquad \rho \vdash \mathrm{E}_2 \to c_2}}{\rho \vdash \mathrm{E}_1\ \mathrm{E}_2 \to push; c_1; swap; c_2; cons; app} \qquad \alpha \cdot \mathrm{s} \vdash push; c_1; swap; c_2; cons; app : \mathrm{s}'}}{(\vec{\rho_1}, \gamma) \cdot \mathrm{s} \vdash c_3 : \delta \cdot \mathrm{s} \text{ (induction)}}}{([\![c_3, \vec{\rho_1}]\!], \gamma) \cdot \mathrm{s} \vdash app : \delta \cdot \mathrm{s}}}{\alpha \cdot \beta \cdot \mathrm{s} \vdash c_2 : \gamma \cdot \beta \cdot \mathrm{s} \text{ (induction)}}}{\alpha \cdot \alpha \cdot \mathrm{s} \vdash c_1 : \beta \cdot \alpha \cdot \mathrm{s} \text{ (induction)}}}{\alpha \cdot \mathrm{s} \vdash push; c_1; swap; c_2; cons; app : \delta \cdot \mathrm{s}}$$

Figure 15. Proof of lemma cam. Rule ml_cam.12.

$$\dfrac{\dfrac{\bar\rho \vdash \mathrm{E}_1 \to c_1 \qquad \dfrac{(\bar\rho_1, \mathrm{P}) \vdash \mathrm{E} \to c}{\bar\rho \vdash \mathrm{E}_2 \to c_2}}{\bar\rho \vdash \mathrm{E}_1\ \mathrm{E}_2 \to push; c_1; swap; c_2; cons; app} \qquad \dfrac{\dfrac{\dfrac{\dfrac{(\vec\rho_1, \beta') \cdot \mathrm{s} \vdash c : \gamma' \cdot \mathrm{s}\ \text{(Hyp.1)}}{(\alpha', \beta') \cdot \mathrm{s} \vdash app : \gamma' \cdot \mathrm{s}}}{\vec\rho \cdot \alpha' \cdot \mathrm{s} \vdash c_2 : \beta' \cdot \alpha' \cdot \mathrm{s}}}{\vec\rho \cdot \vec\rho \cdot \mathrm{s} \vdash c_1 : \alpha' \cdot \vec\rho \cdot \mathrm{s}}}{\vec\rho \cdot \mathrm{s} \vdash push; c_1; swap; c_2; cons; app : \gamma' \cdot \mathrm{s}}}{\dfrac{\rho \vdash \mathrm{E}_2 : \beta \qquad \vdash \beta \to \beta'\ \text{(ind.)} \qquad \rho \vdash \mathrm{E}_1 : \alpha = [\![\lambda \mathrm{P.E}, \rho_1]\!] \qquad \vdash \alpha \to \alpha'\ \text{(ind. \& Hyp.1)}}{\mathrm{P} \mapsto \beta \cdot \rho_1 \vdash \mathrm{E} : \gamma \qquad \vdash \gamma \to \gamma'\ \text{(ind.)}}}{\rho \vdash \mathrm{E}_1\ \mathrm{E}_2 : \gamma \qquad \vdash \gamma \to \gamma'}$$

Figure 16. **Rule ml_cam.12:** $e = \mathrm{E}_1\ \mathrm{E}_2$, 1st case.

$$\dfrac{\dfrac{\bar\rho \vdash \mathrm{E}_1 \to c_1 \qquad \bar\rho \vdash \mathrm{E}_2 \to c_2}{\bar\rho \vdash \mathrm{E}_1\ \mathrm{E}_2 \to push; c_1; swap; c_2; cons; app} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{\text{eval}}{\vdash}\ top, \gamma_1', \gamma_2' : \gamma'\ \text{(Hyp.2)}}{(0, \beta') \cdot \mathrm{s} \vdash cdr; top : \gamma' \cdot \mathrm{s}\ \text{(Hyp.2)}}}{(\alpha', \beta') \cdot \mathrm{s} \vdash app : \gamma' \cdot \mathrm{s}}}{\vec\rho \cdot \alpha' \cdot \mathrm{s} \vdash c_2 : \beta' \cdot \alpha' \cdot \mathrm{s}}}{\vec\rho \cdot \vec\rho \cdot \mathrm{s} \vdash c_1 : \alpha' \cdot \vec\rho \cdot \mathrm{s}}}{\vec\rho \cdot \mathrm{s} \vdash push; c_1; swap; c_2; cons; app : \gamma' \cdot \mathrm{s}}}{\dfrac{\rho \vdash \mathrm{E}_2 : \beta \qquad \vdash \beta \to \beta'\ \text{(ind.)} \qquad \rho \vdash \mathrm{E}_1 : \alpha = \mathrm{ident\,OP} \qquad \vdash \alpha \to \alpha'\ \text{(ind. \& Hyp.2)}}{\overset{\text{eval}}{\vdash}\ \mathrm{OP}, \gamma_1, \gamma_2 : \gamma \qquad \vdash \gamma_1 \to \gamma_1' \qquad \vdash \gamma_2 \to \gamma_2' \qquad \vdash \gamma \to \gamma'\ (\text{lemma "}eval_2\text{"})}}{\rho \vdash \mathrm{E}_1\ \mathrm{E}_2 : \gamma \qquad \vdash \gamma \to \gamma'}$$

Figure 17. **Rule ml_cam.12:** $e = \mathrm{E}_1\ \mathrm{E}_2$, 2nd case.

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{x}' = v_1' \qquad (\vec\rho, v_1') \cdot \mathrm{s} \vdash c_2 : \beta' \cdot \mathrm{s}}{(\vec\rho, \mathrm{x}') \cdot v_1' \cdot \mathrm{s} \vdash rplac : (\vec\rho, v_1') \cdot \mathrm{s}}}{(\vec\rho, \mathrm{x}') \cdot (\vec\rho, \mathrm{x}') \cdot \mathrm{s} \vdash c_1 : v_1' \cdot (\vec\rho, \mathrm{x}') \cdot \mathrm{s}}}{\vec\rho \cdot \mathrm{s} \vdash push; quote(\mathrm{x}'); cons; push; c_1; swap; rplac; c_2 : \beta' \cdot \mathrm{s}}$$

$$\dfrac{\dfrac{(\bar\rho, \mathrm{P}) \vdash \mathrm{E}_1 \to c_1 \qquad (\bar\rho, \mathrm{P}) \vdash \mathrm{E}_2 \to c_2}{\bar\rho \vdash \text{letrec P} = \mathrm{E}_1 \text{ in } \mathrm{E}_2 \to push; quote(\mathrm{x}'); cons; push; c_1; swap; rplac; c_2}}{\dfrac{\dfrac{\mathrm{P} \mapsto \mathrm{x} \cdot \rho \vdash \mathrm{E}_1 : v_1 \qquad \vdash \mathrm{x} \to \mathrm{x}' \qquad \vdash v_1 \to v_1'\ \text{(ind.)}}{\mathrm{P} \mapsto v_1 \cdot \rho \vdash \mathrm{E}_2 : \beta \qquad \vdash \beta \to \beta'\ \text{(ind.)}}}{\rho' = \mathrm{P} \mapsto [\![\mathrm{E}_1, \rho']\!] \cdot \rho \vdash \mathrm{E}_2 : \beta}\ (lemma\lambda_2)}{\rho \vdash \text{letrec P} = \mathrm{E}_1 \text{ in } \mathrm{E}_2 : \beta \qquad \vdash \beta \to \beta'}$$

Figure 18. **Rule ml_cam.9:** $e = \text{letrec P} = \mathrm{E}_1 \text{ in } \mathrm{E}_2$.

Rule ml_cam.12: $e = \mathrm{E}_1\ \mathrm{E}_2$, 1st case. The proof tree is given in Fig. 16. To draw it, we made one hypothesis on the form that can have α' for making the apply rule in cam_ds applicable. Hyp.1: $\alpha' = [\![c, v(\rho_1)]\!] \Rightarrow \alpha = [\![\lambda \mathrm{P.E}, \rho_1]\!]$.

Rule ml_cam.12: $e = \mathrm{E}_1\ \mathrm{E}_2$, 2nd case. The alternative hypothesis says that α' is a closure again, but translation of an identifier. Hyp.2: $\alpha' = [\![cdr; top, 0]\!] \Rightarrow \alpha = \mathrm{ident\,OP} \Rightarrow \beta' = (\gamma_1', \gamma_2')$. We use here a second lemma on eval, similar to the previous one, and taken as hypothesis as well:

Lemma $eval_2$.

$$\exists \alpha, \beta, \gamma \quad \dfrac{\overset{\text{trans\_const}}{\vdash}\ \mathrm{ident\,OP} \to c \qquad \overset{\text{eval}}{\vdash}\ c, \alpha', \beta' : \gamma'}{\overset{\text{eval}}{\vdash}\ \mathrm{OP}, \alpha, \beta : \gamma \vdash \alpha \to \alpha' \vdash \beta \to \beta' \vdash \gamma \to \gamma'}$$

Then we can draw our proof tree (see Fig. 17).

Rule ml_cam.9: e = letrec P = E_1 in E_2. We need here a lemma on λ-exp. and list of closures. The intuitive meaning of this very technical lemma will be clear, we hope so, by its proof, and by the proof tree it allows to draw.

Lemma λ_2. For E_1 being a list of λ-expressions, we have:

$$\frac{\begin{array}{c}P \mapsto X \cdot \rho \vdash E_1 : v_1 \qquad (\bar{\rho}, P) \vdash E_1 \to c_1 \\ (\vec{\rho}, t(X)) \cdot s \vdash c_1 : t(v_1) \cdot s \qquad t(v_1) = t(X) \\ P \mapsto v_1 \cdot \rho \vdash E_2 : \beta\end{array}}{\rho' = P \mapsto [\![E_1, \rho']\!] \cdot \rho \vdash E_2 : \beta}$$

Proof. The proof is easy, using the three technical facts we just state bellow:

- If $\rho \vdash (e_i)_i : v$ with $(e_i)_i \in \lambda$-exp. then $v = ([\![e_i, \rho]\!])_i$
- If $t(v) = t(([\![e_i, \rho]\!])_i)$ then $v = ([\![e_i, \rho]\!])_i$ or $v = [\![(e_i)_i, \rho]\!]$
- if $(\bar{\rho}, P) \vdash (e_i)_i \to c$ and $(\vec{\rho}, t(X)) \cdot s \vdash c : t(v) \cdot s$ with $(e_i)_i \in \lambda$-exp. then $t(v) = t(([\![e_i, P \mapsto X \cdot \rho]\!]))_i$

□ *Lemma* λ_2

Now we can draw our proof tree (see Fig. 18).

□ Rule (2)

8. Conclusion

Semantic definitions appear to be very compact, thanks to a very general style, and thanks to some extra possibilities, such as the use of graphs, for example, which enables us to give a very clear and compact semantics for the *letrec* construct. Translation specification is written in the same style as static semantics. Semantic definitions deal with validity of theorems ($\rho \vdash e : \alpha$, $\rho \vdash e \to c$ ...) in formal systems, and proofs of translation deal with validity of inference rules in the union of these formal systems.

The device of using the union of several formal systems is not only interesting to formalize a proof, it appear to provide a good framework for formalizing mixed execution (i.e. execution of partially compiled programs). More precisely, once the correctness of a translation has been proved, we can add the just proved inference rules in $\mathcal{T}$. Now, suppose we have compiled some parts of a program (using the semantic definition of the translation). The execution of the mixed program obtained is specified by the dynamic semantics of the source language, the semantics of the machine code, together with the new rules which appear to be 'switching rules' from a language to another. In particular, the new rules explain how to communicate environments between interpreted and compiled code. For the moment, this actually works when the translation on semantic values is a one-one mapping. Experiments have been made with a small Pascal-like language.

References

[1] [LNCSn stands for Vol n, Lecture Notes in Computer Science, Springer-Verlag].

[2] J. BARWISE, "The Handbook of Mathematical Logic", Nortl Holland, Amsterdam, 1977, reprinted in 1983.

[3] G. COUSINEAU, P. L. CURIEN, M. MAUNY, "The categorical Abstract machine", LITP Report 85-8, University Paris VII, january 1985. also Proc. of the IFIP conference on "Functional Programming Languages and Computer Architecture", Nancy, France, Sept. 1985, LNCS 201.

[4] T. DESPEYROUX, "Executable Specification of Static Semantics", Semantics of Data Types, 1984, LNCS 173.

[5] T. DESPEYROUX, "Typol: a formalism to implement natural semantics", to appear.

[6] G. GENTZEN, "The Collected Papers of Gerhard Gentzen", E. Szabo, North-Holland, Amsterdam, 1969.

[7] G. HUET, "Deduction and Computation", INRIA course "Methods and languages for artificial intelligence", November 1985.

[8] D. CLEMENT, J. DESPEYROUX, T. DESPEYROUX, L. HASCOET, G. KAHN, "Natural semantics on the computer", I.N.R.I.A Report RR 416, June 85.

[9] W. LI, "An operational approach to semantics and translation for concurent programming languages", Ph.D., Edinburgh, 1983.

[10] J.J. LEVY, "Réductions sures dans le lambda-calcul", Thèse de 3ième cycle, Paris VII, France, 1974.

[11] M.J. GORDON, A.J. MILNER, CHRISTOPHER P. WADSWORTH, "Edinburgh LCF", LNCS 78, 1979.

[12] M. MAUNY, "Compilation des langages fonctionnels dans les combinateurs catégoriques. Application au langage ML.", Thèse de 3ième cycle, Paris VII, France, 1985.

[13] D. CLEMENT, J. DESPEYROUX, T. DESPEYROUX, G. KAHN, "A Simple Applicative Language: Mini-ML", Submitted for publication.

[14] F.L. MORRIS, "Advice on structuring compilers and proving them correct", Principles Of Programming Languages, 1973.

[15] V. DONZEAU-GOUGE, G. KAHN, B. LANG, B. MELESE "Document stucture and modularity in Mentor", Proc. ACM SIGSOFT/SIGPLAN Soft. Eng. Symp. on Practical Software development Environments, Pittsburgh, april 1984.

[16] G.D. PLOTKIN, "A structural approach to operational semantics", Aarhus Report DAIMI FN-19, 1981.

[17] G.D. PLOTKIN, "A powerdomain for Countable Non Determinism", Proc. of the ICALP conference, Aarhus, Denmark, 1982, LNCS 140.

[18] D. PRAWITZ, "Natural Deduction, a Proof-Theoretical Study", Almqvist & Wiksell, Stockholm, 1965.

[19] C.P. WADSWORTH, "The relation between computational and denotational properties for Scott's D. - Models of the lambda calculus", Siam Journal on Computing, 1976, Vol. 5.